Question1

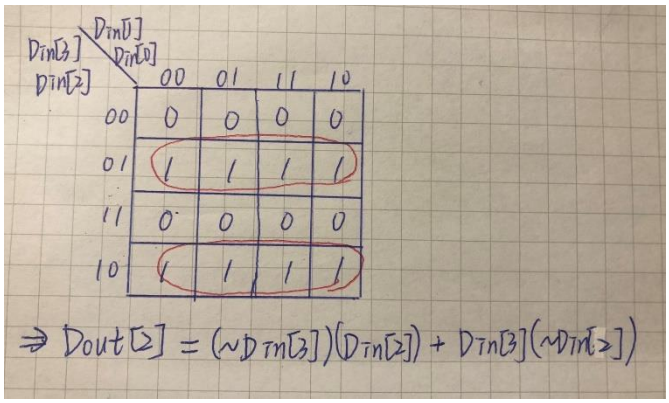
■ Din[3:0] = Binary code; Dout[3:0] = Grey code

Decimal value	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Decimal value	Binary code	Gray code
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

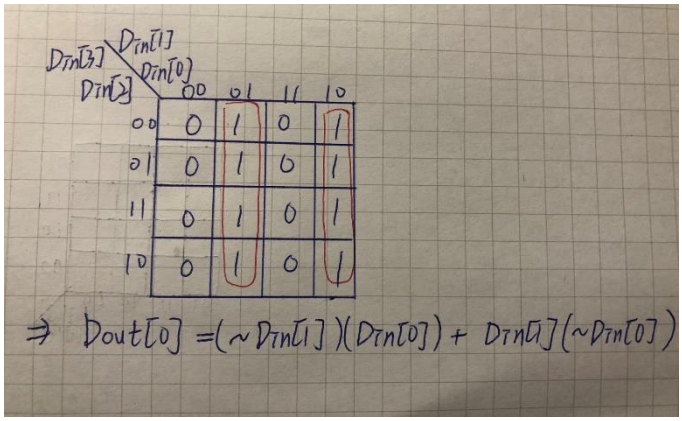
由題幹給的表格即得知其 truth table，首先，觀察 Dout[3]，可發現在每一個數字時，Dout[3]皆與 Din[3]相等，因此可以得到 Logic equation : Dout[3] = Din[3]。

針對 Dout[2]與 Din[2]的關係，我們可以用 K-map 化簡，以下為 K-map 的化簡過程：



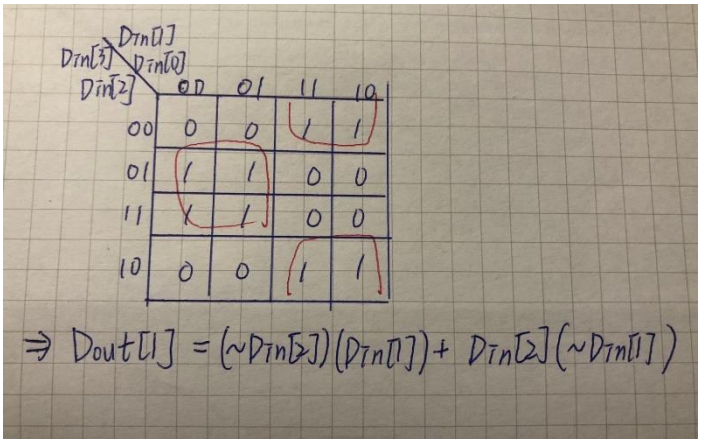
由此得到：Dout[2] = (~Din[3])Din[2] + Din[3](~Din[2]) = Din[3]⊕Din[2]　(XOR)

同理，針對 Dout[1]與 Din[1]的關係，一樣用 K-map 化簡，以下為 K-map 的化簡過程：



由此得到：Dout[1] = (~Din[2])Din[1] + Din[2](~Din[1]) = Din[2]⊕Din[1]

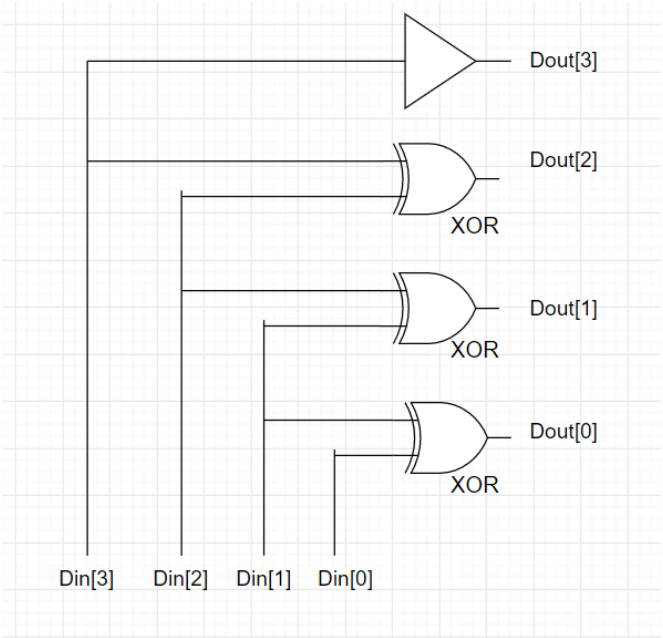
同理，針對 Dout[0]與 Din[0]的關係，一樣用 K-map 化簡，以下為 K-map 的化簡過程：

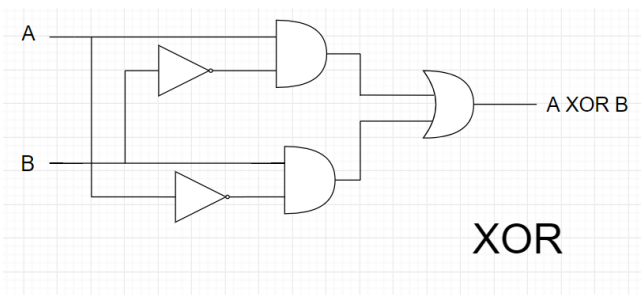


由此得到：Dout[0] = (~Din[1])Din[0] + Din[1](~Din[0]) = Din[1]⊕Din[0]

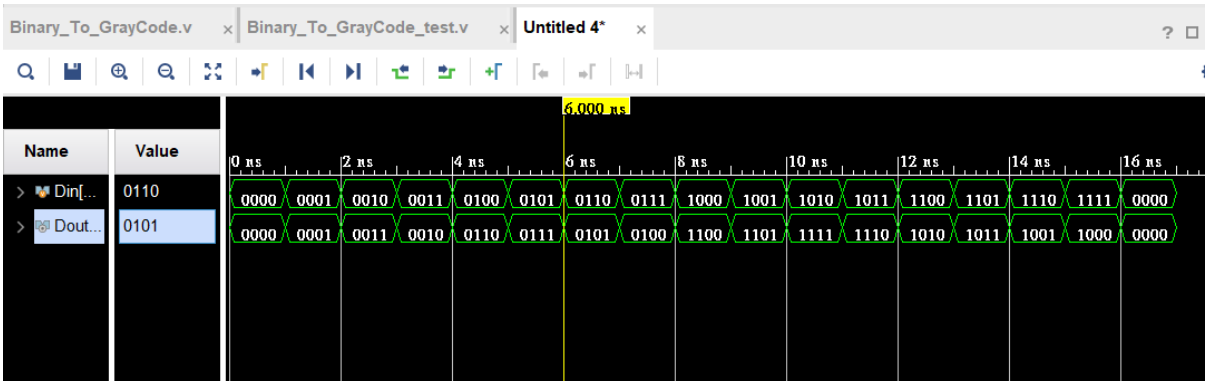
根據以上所得到的 Logic equation：
Dout[3] = Din[3]
Dout[2] = Din[3]⊕Din[2]
Dout[1] = Din[2]⊕Din[1]
Dout[0] = Din[1]⊕Din[0]

可畫出以下的電路圖：





由 Vivado 可模擬出以下的波形圖：



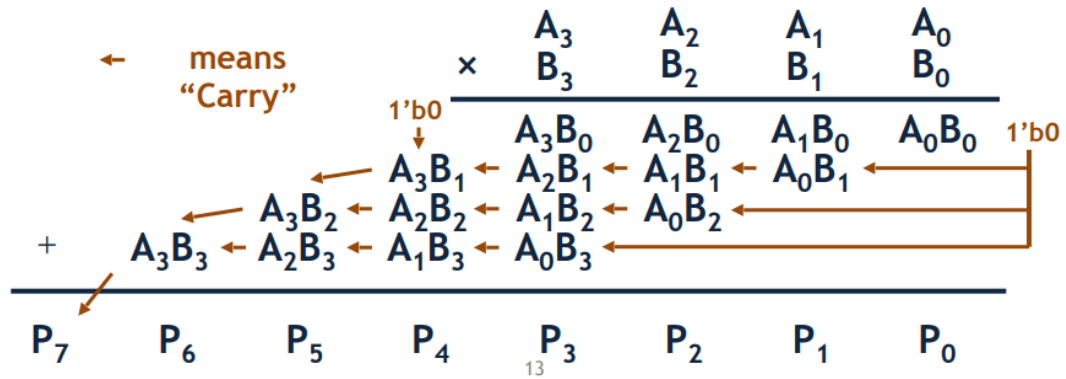
由上方的波形模擬可證實 16 個 Binary code 確實有分別對應到其 Grey code，以上 simulation results 皆符合預期。

Testbench 設計理念：

因為此題的 output 只有一個而且僅有 4bits，產生出的結果只有 16 種情形，因此採用每次迴圈餵 1-bit 的方式，最後觀察結果是否符合預期。

在 testbench 裡中，預設 Din = 0000，在每一次的迴圈中都加上 4’ b0001，總共執行了 2 的 4 次方次(會從 4’ b0000 加到 4’ b1111)，如此一來就能檢測到所有的可能的測資。

Question2



根據題目給的這張表，大致可以了解乘法器如何運作：

首先，觀察第一位數(A[0]B[0])，得到 P[0] = A[0]B[0]。

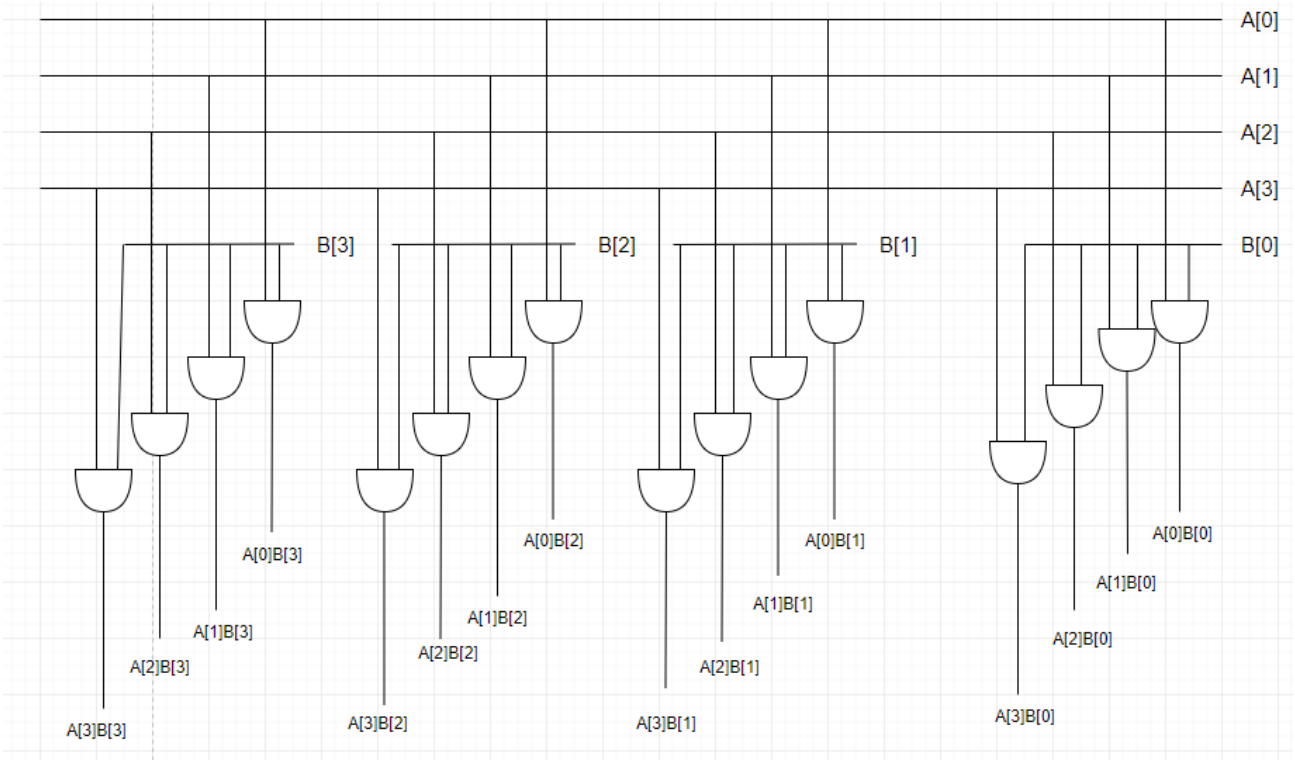
第二位數為 A[1]B[0] + A[0]B[1] + 1’b0 ，利用 1-bit Full Adder 將此三數加起來得到 P[1]以及進位數到下一位數 C[1]。

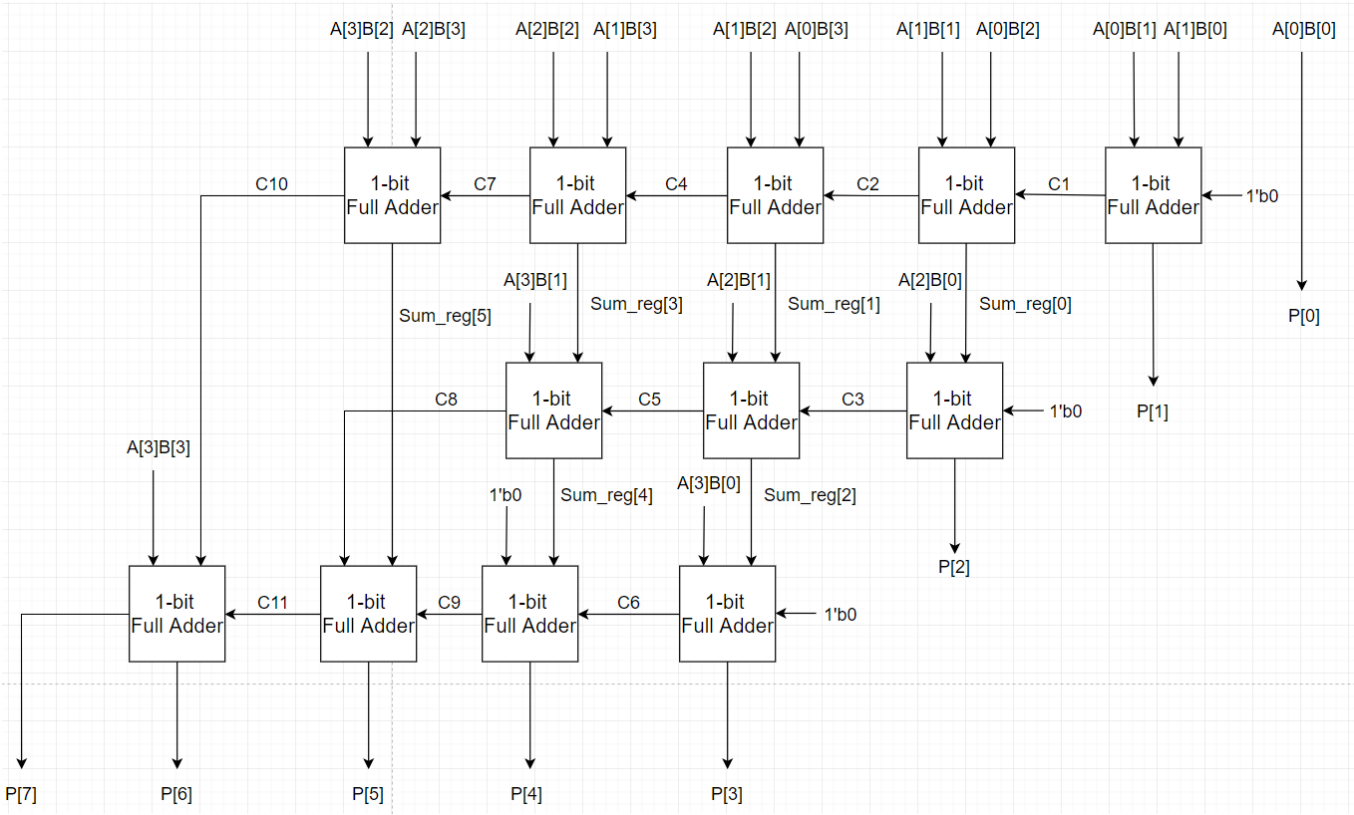
第三位數則要用兩個 1-bit Full Adder 加起來，第一個 Full Adder：A[0]B[2] + A[1]B[1] + C[1]，得到一個和先暫存在 Sum_reg[0]，還得到第一個進位數到下一位 C[2]。第二個 Full Adder：A[2]B[0] + Sum_reg[0] + 1'b0，得到 P[2]以及第二個進位數到下一位 C[3]。

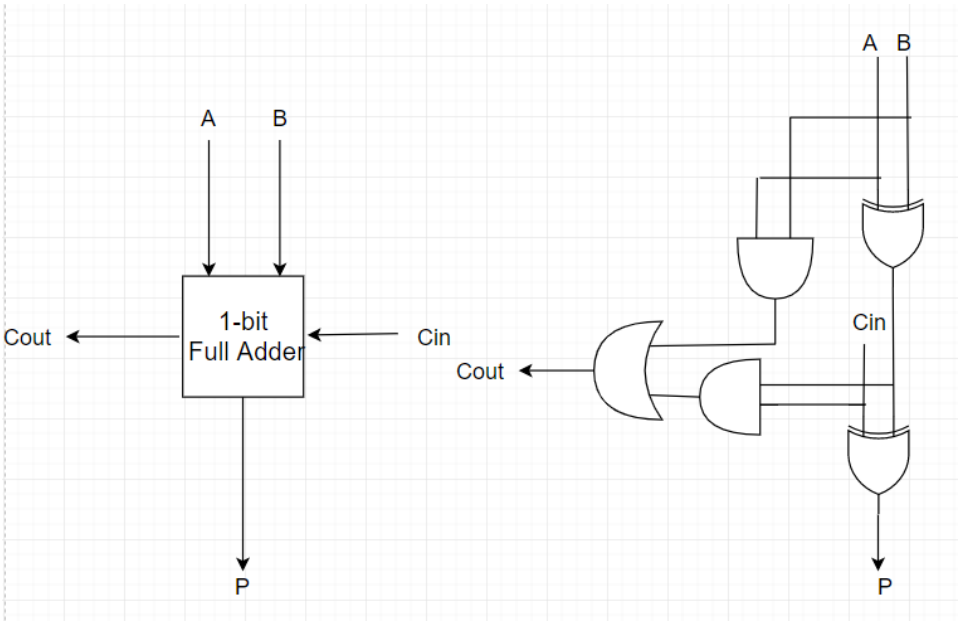
第四位數因為有 6 個東西要相加(分別為 A[0]B[3], A[1]B[2], A[2]B[1], A[3]B[0], C[2]以及 C[3])，會需要用到三個 Full Adder 加起來。第一個 Full Adder：A[0]B[3] + A[1]B[2] + C[2]，得到和暫存到 Sum_reg[1]，還有第一個進位數 C[4]到下一位。第二個 Full Adder：A[2]B[1] + A[3]B[0] + C[3]，得到和暫存到 Sum_reg[2]，還有第二個進位數 C[5]到下一位。第三個 Full Adder：Sum_reg[1] + Sum_reg[2] + 1'b0，得到和 P[3]，還有第三個進位數 C[5]到下一位。

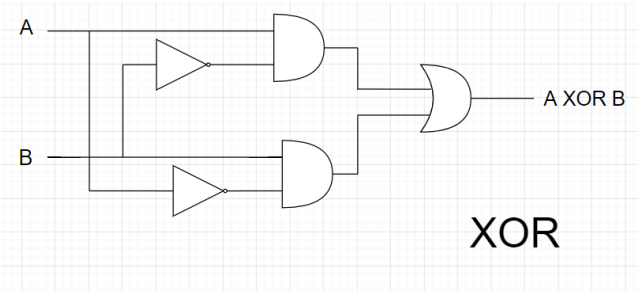
其他位數以此類推……。

因此由以上邏輯可以畫出以下的電路圖：

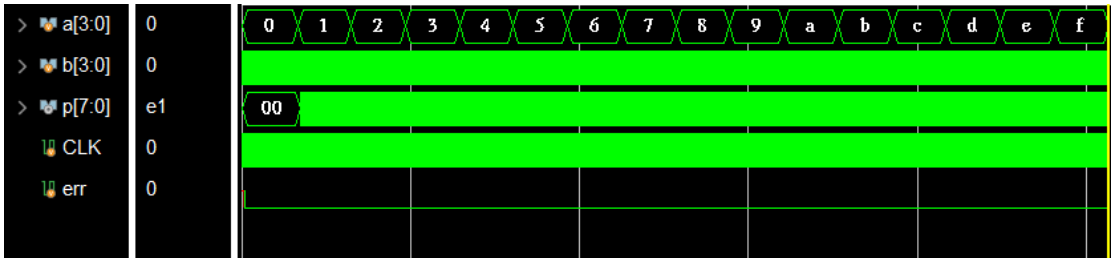



A[3]B[2]
A[2]B[3]
A[2]B[2]
A[1]B[3]
A[1]B[2]
A[0]B[3]
A[1]B[1]
A[0]B[2]
A[0]B[1]
A[1]B[0]
A[0]B[0]
1-bit Full Adder
C10
C7
C4
C2
C1
1'b0
P[0]
A[3]B[1]
A[2]B[1]
A[2]B[0]
Sum_reg[5]
Sum_reg[3]
Sum_reg[1]
Sum_reg[0]
C8
C5
C3
P[1]
A[3]B[3]
1'b0
Sum_reg[4]
A[3]B[0]
Sum_reg[2]
P[2]
C11
C9
C6
P[7]
P[6]
P[5]
P[4]
P[3]


A
B
1-bit Full Adder
Cout
Cin
P
A
B
Cin
Cout
P


A
B
A XOR B
XOR

由 Vivado 可模擬出以下的波形圖：



Verification
After drawing the logic circuit, we use verilog to realize the circuit we design and verify that whether the output is the result we want. If the output is not the thing we expect, it would output an error message 1.

By the diagram above, it proves that the logic circuit and verilog code are correct.

Question3

由 Lab1 所做出的 Ripple Carry Adder 有用到 Full Adder 的 logic equation：

$C_{i+1} = A_iB_i + B_iC_i + C_iA_i = A_iB_i + (A_i \oplus B_i)\, C_i = G_i + P_iC_i \quad (G_i = A_iB_i\,,\ P_i = A_i \oplus B_i)$

$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$

以下列舉情形：

$C_1 = G_0 + P_0C_0$;

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$;

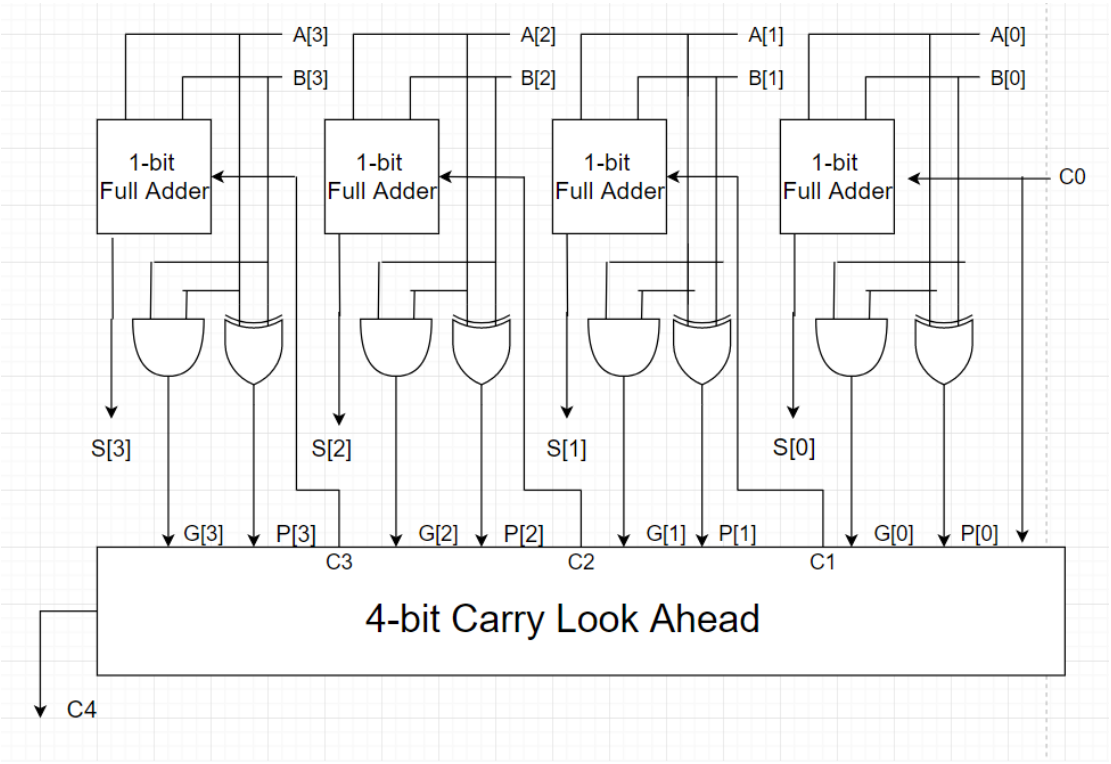
$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$;

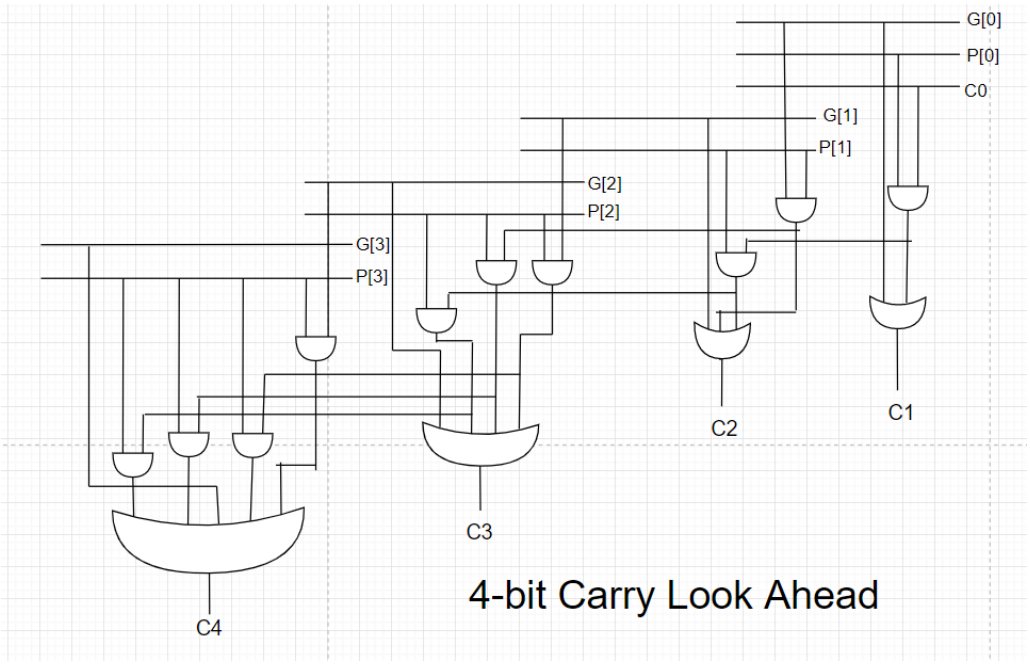
$C_4 = G_3 + P_3C_3 = G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$

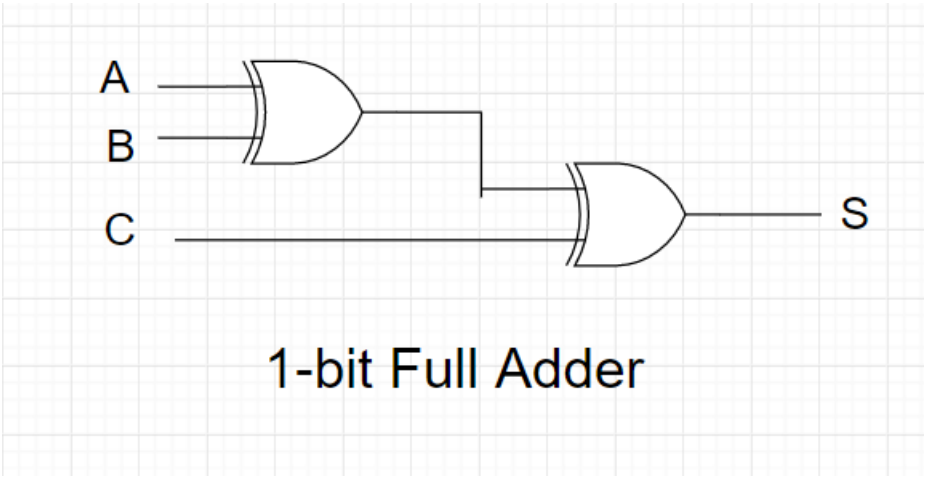
$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$;

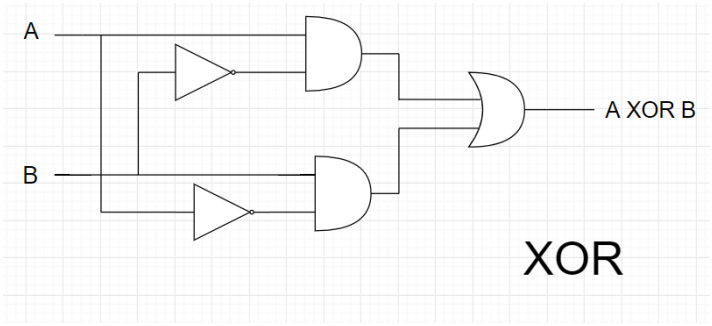
當 input 有給的情況下(namely A_i & B_i)，G_i 以及 P_i 皆屬於已知數($G_i = A_iB_i$, $P_i = A_i \oplus B_i$)，且由以上列舉情形可發現，C_1,C_2,C_3,C_4 經過代換後，都只跟 C_0, G_i 以及 P_i 有關。根據以上的發現可以得知，Carry Look Ahead Adder 的優點為：無論是哪一位數的數字，皆無需等待進位數即可做運算，因此跟 Lab1 做的 Ripple Carry Adder 相比，Carry Look Ahead Adder 所需的時間會比較短。

由以上所得到的 logic equation 可以畫出以下的電路圖：


A[3]
B[3]
A[2]
B[2]
A[1]
B[1]
A[0]
B[0]
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
1-bit Full Adder
C0
S[3]
S[2]
S[1]
S[0]
G[3]
P[3]
G[2]
P[2]
G[1]
P[1]
G[0]
P[0]
C3
C2
C1
4-bit Carry Look Ahead
C4


G[0]
P[0]
C0
G[1]
P[1]
G[2]
P[2]
G[3]
P[3]
C2
C1
C3
C4
4-bit Carry Look Ahead





電路設計理念：

一開始先將 A[0], B[0], A[1], B[1], A[2], B[2], A[3], B[3]先拉出來將個別的 P[0], G[0], P[1], G[1], P[2], G[2], P[3], G[3]做出來，接下來將它們同時送進 4-bit Carry Look Ahead Adder 做出所有的進位數，包含 C1, C2, C3, C4，再將 C1, C2, C3 分別送到對應的 Full Adder 算出該位數的和，分別包含 S[1], S[2], S[3](至於 S[0]因為一開始 input 就有 C0 所以算出來了)，以上的電路設計圖是分區塊模組寫出來的。

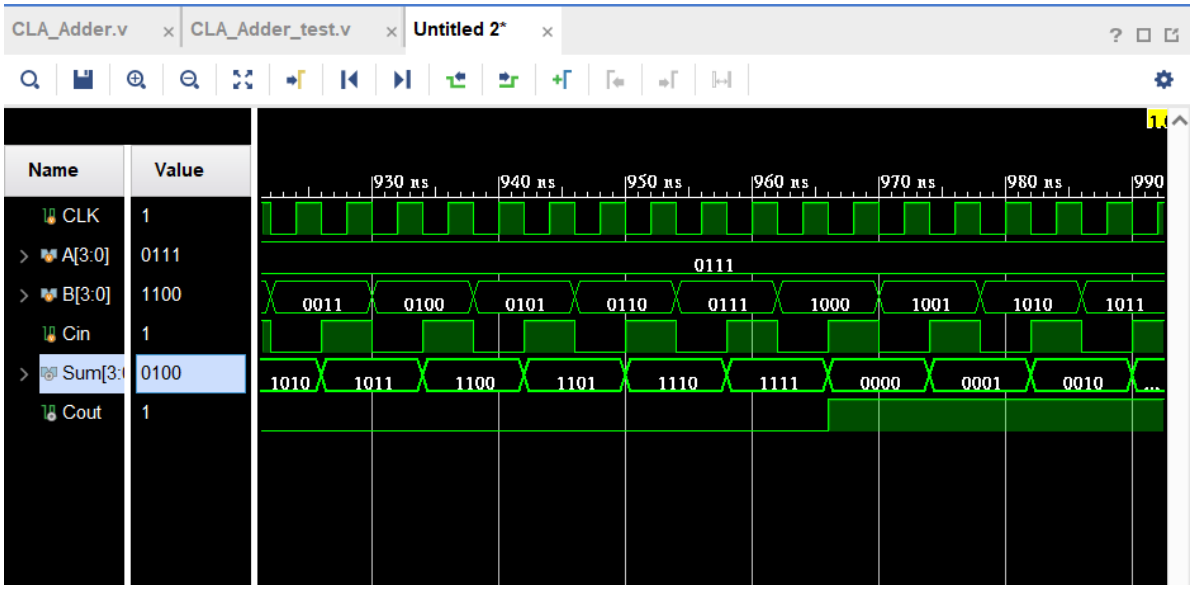
Testbench 設計理念：

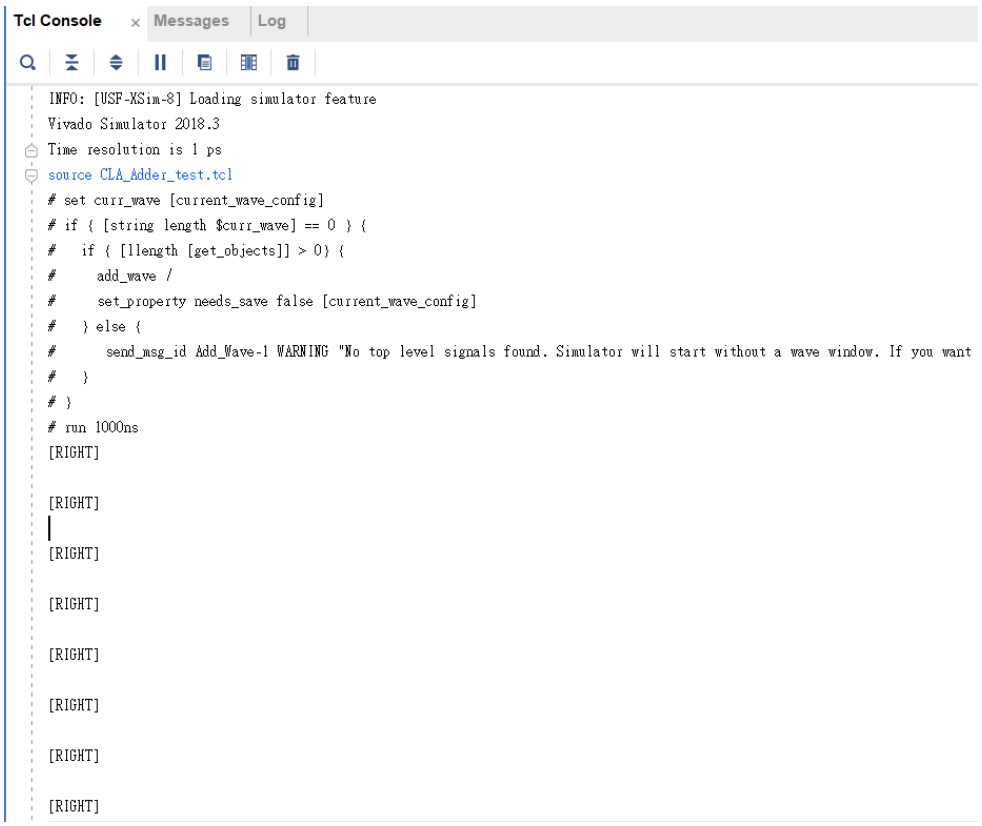
在 testbench 中，一開始預設{A, B, C0} = 9'd0，此外多令一個變數為 CLK，一開始 CLK = 1'b1，且每經過 5 ns，CLK 會接一個 invertor，形成週期為 5 ns 的 clock。

預設當 CLK = 1'b1 時，就會呼叫一個 function 去檢查是否{C4, S}會等於 (A + B + C0)，如果不相等，則顯示[ERROR]，且將 input 以及 output 全部分別印出來；如果相等，即顯示[RIGHT] ，如此一來會比較好 debug。

然而，當 CLK = 1'b0 時，會餵 1-bit 給{A, B, Cin} ，也就是{A, B, C4} = {A, B, C4} + 1'b1，連續重複 2 的 9 次方次，如此一來，就能檢測到所有的測資。

由 Vivado 可模擬出以下的波形圖：





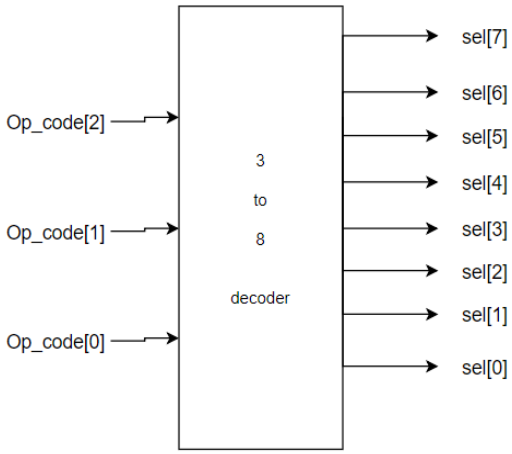
因為這題的 Case 太多了，單看波形模擬圖可能會沒有辦法檢查完所有結果，可以看波形模擬圖下方的 Tcl Console 那一欄的資訊，如果所有的答案都是對的，根據 testbench 上寫的，錯誤的會顯示[ERROR]；正確的會顯示[RIGHT]。如上圖可知，所有答案皆符合測資，以上 simulation results 皆符合預期。

Question4

Input [2:0] Op_code;
Input [3:0] Rs, Rt;
Output [4:0] Rd;

Design Process
(first)

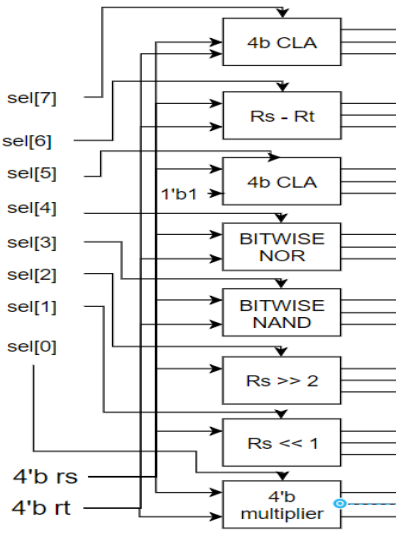
首先設計一個 3 to 8 的 decoder, 當特定的 Op_code 輸入時，對應

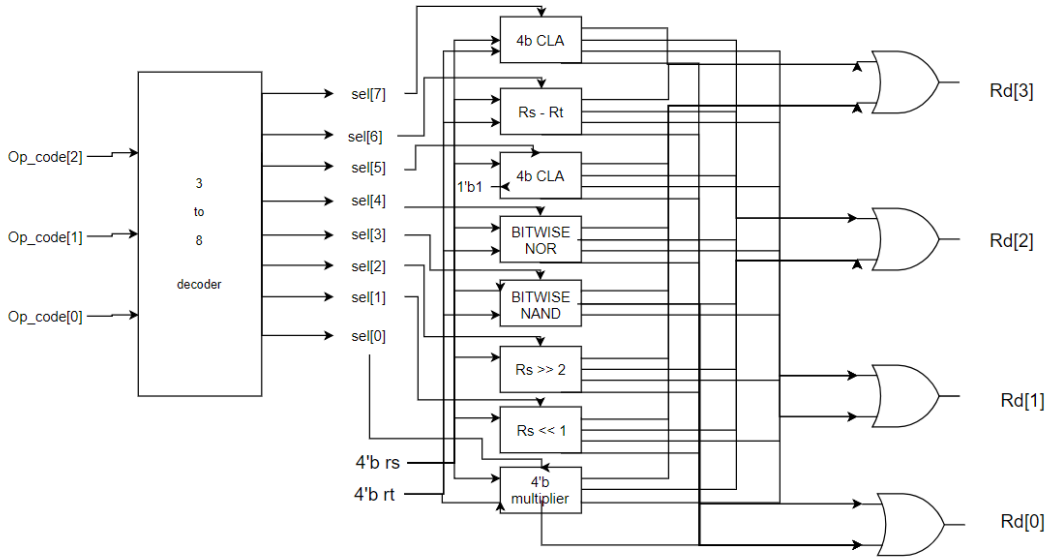


到該 OP_code 的 sel 訊號才會為 1，其餘情況為 0

(second step)

設計 8 個 mux，分別把 8 種 rs 與 rt 的結果運算出來，然後再與對應該 Op_code 的 sel 訊號做 and 的運算，此步驟的意義即保留對應到該 OP_code 的運算數值，其餘丟棄。

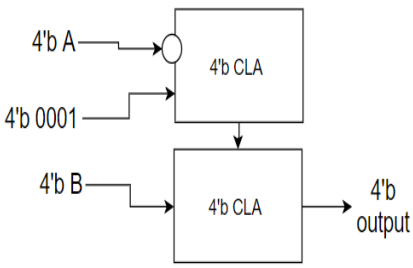




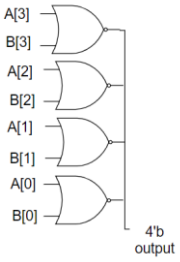
(third step)

最後再將每一位數值做 or 運算，此步驟的意義為將上一步運算保留的數值輸出，下圖為完整的邏輯電路。

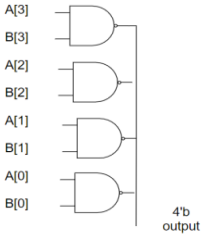
From lab2-2 and lab2-3, we get the logic circuit of multiplier and carry look ahead adder.



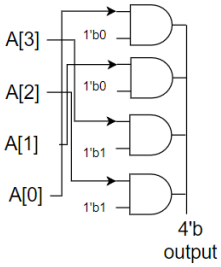
module rs – rt

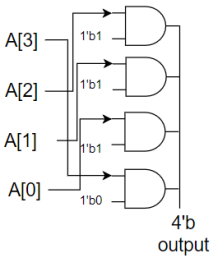


module BITWISE NOR



module BITWISE NAND

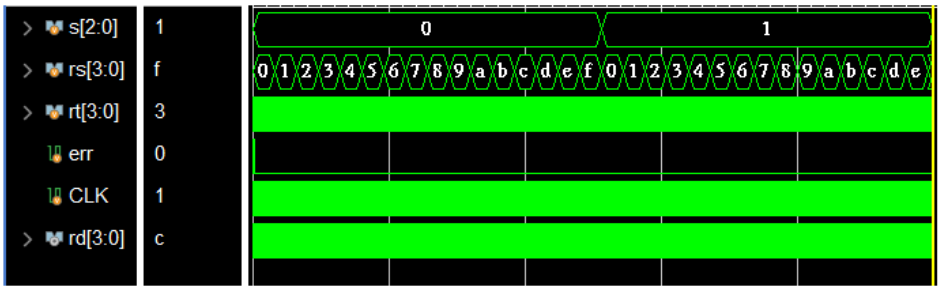


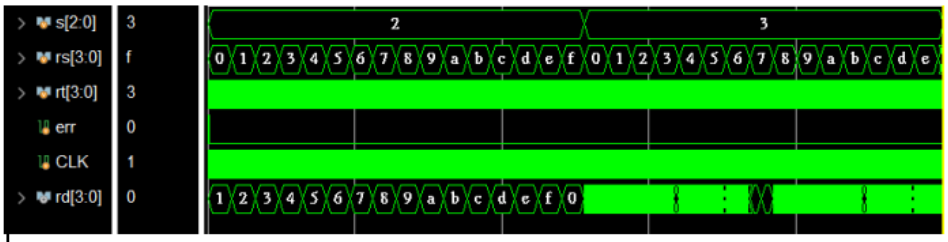


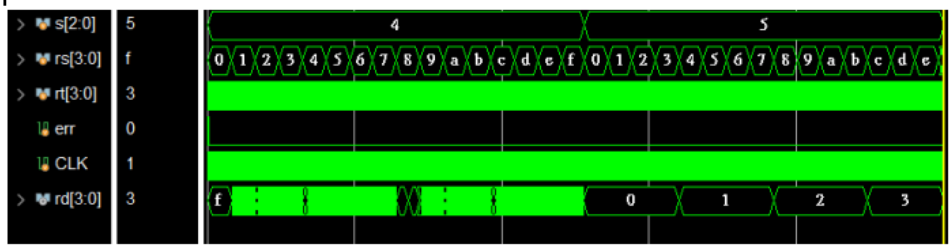
Module rs << 2 module rs >> 1

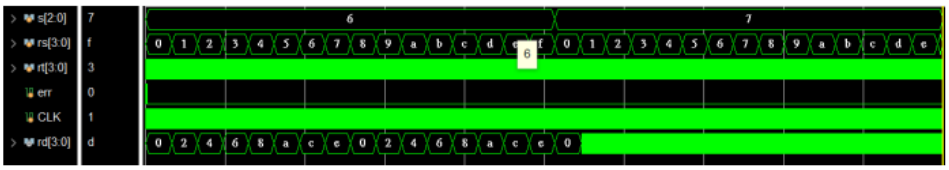
Verification

After drawing the logic circuit, we use verilog to realize the circuit we design and verify that whether the output is the result we want. If the output is not the thing we expect, it would output an error message 1.

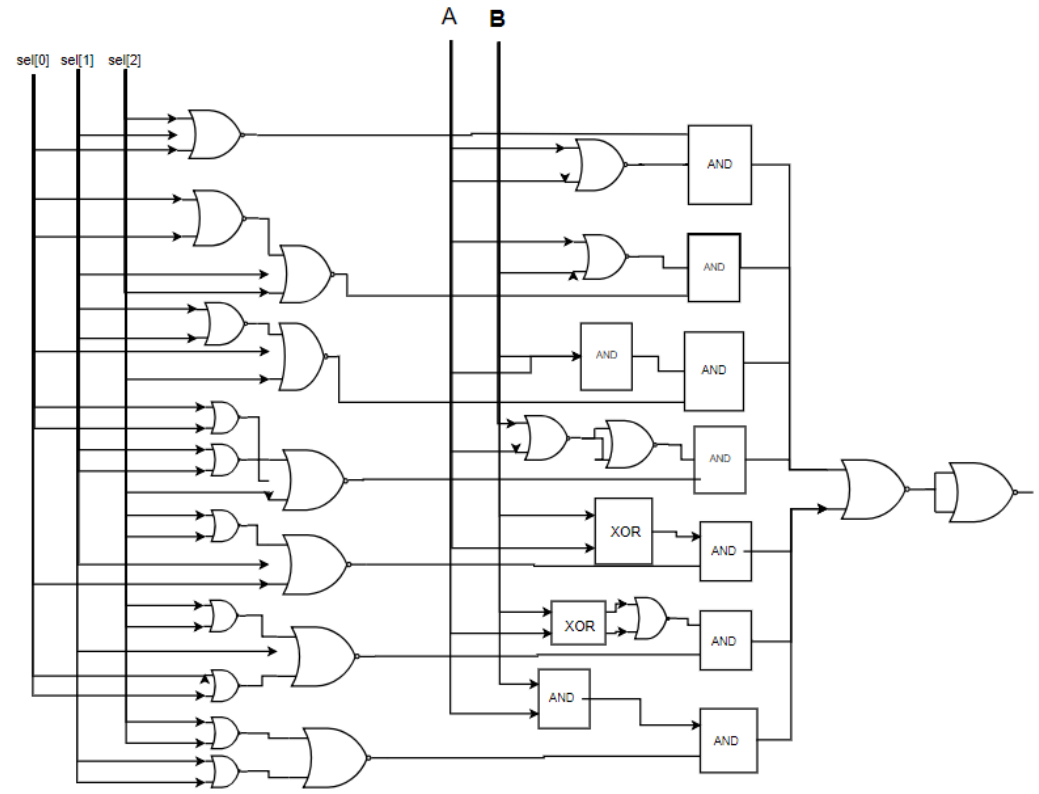


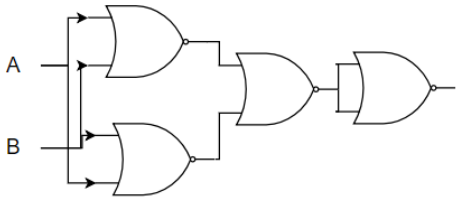




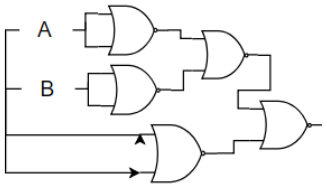


By the diagram above, it proves that the logic circuit and verilog code are correct.


A
B
sel[0]
sel[1]
sel[2]
AND
AND
AND
AND
AND
XOR
AND
XOR
AND
AND
AND

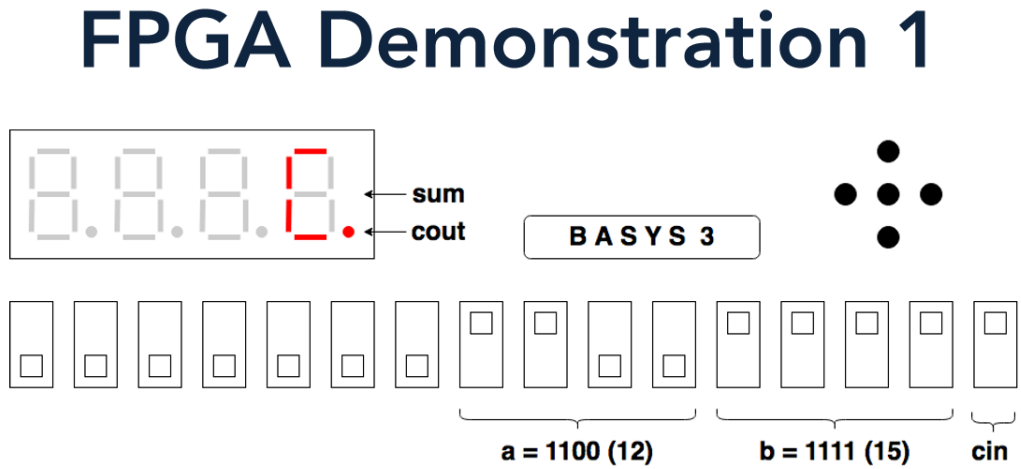

A
B

AND


A
B

XOR

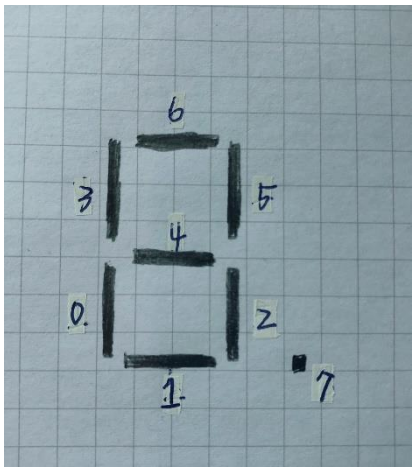
FPGA demonstration



第三題的 input 為[3:0] A, [3:0] B 以及 C0; output 為[3:0] S 以及 C4，在 input 端[3:0] A, [3:0] B 以及 C0 要接到 FPGA 板的 Switch 端共 9 個接腳(如上圖)。

至於七段顯示器中，因為有 4 個數字可以顯示亮度，而每個位數的數字共有 8 個腳位可以去控制亮暗與否(包含 7 條線與 1 個小數點)。此題因為只要求第一個數字亮，其他三個數字都要暗，所以我令一個變數[3:0] Dark，Dark 的每一個 bit 接到相對應的數字(Dark[3]接第四個數字、Dark[2]接第三個數字、Dark[1]接第二個數字、Dark[0]接第一個數字)並 assign Dark = 4'b1110 (七段顯示器接腳接 1 會暗; 接 0 會亮)。

接下來，因為一個數字裡面，總共會有 8 個腳位能控制亮暗，所以代表會有 8 個 output 要接上七段顯示器中一個數字的 8 個腳位，所以要將原本 output[3:0] S 改成 wire [3:0] S，並多另一個 output [7:0]dec 去連接那 8 個腳位。此時做一個 decoder 去將{C4, S}的所有情形(5bits，所以共有 32 種情形)編碼，假設今天要顯示數字 1 在七段顯示器上，以下圖為例(下圖中的每一個腳位上面標示的數字代表第幾個 bit)，dec = 1_1011011(注意：接 1 代表暗；接 0 代表亮)；假設要顯示 0 的話，dec = 1_0010000。



```
module Decoder(cout, sum, out);
input [3:0] sum;
input cout;
output [7:0] out;
reg [7:0] out;
always @*
    case({cout, sum})
        5'd0 : out = 8'b1_0010000;
        5'd1 : out = 8'b1_1011011;
        5'd2 : out = 8'b1_0001100;
        5'd3 : out = 8'b1_0001001;
        5'd4 : out = 8'b1_1000011;
        5'd5 : out = 8'b1_0100001;
        5'd6 : out = 8'b1_0100000;
        5'd7 : out = 8'b1_0010011;
        5'd8 : out = 8'b1_0000000;
        5'd9 : out = 8'b1_0000001;
        5'd10 : out = 8'b1_0000010;
        5'd11 : out = 8'b1_1100000;
        5'd12 : out = 8'b1_0110100;
        5'd13 : out = 8'b1_1001000;
        5'd14 : out = 8'b1_0100100;
        5'd15 : out = 8'b1_0100110;

        5'd16 : out = 8'b0_0010000;
        5'd17 : out = 8'b0_1011011;
```

左圖是我個別編碼的 code

(我是自己做一個 Decoder 的 module，然後在 top module 中呼叫 Decoder，除了這個 module，還有 top module 中的 assign Dark = 4'b1110 是用 Data flow modeling 以及 Behavioral-Modeling，其他全部都是用 Gate level 表示！)

心得

這次的 Lab 雖然沒有說到很難，但是我們卻花了不少時間去 debug、弄 FPGA 板以及用電腦畫電路圖，其中電腦畫電路圖的花很久的時間，可能是還不太會用畫圖軟體。這次的 Lab 讓我學到不少 debug 的技巧、對燒板子的流程比較熟練以及比較熟悉 verilog 的語法，希望這次的進步能為下次的 Lab 減少一點時間，我好想睡覺 QQ。(107060011 涂皓鈞)

小組分工

涂皓鈞 － 負責 Question 2、3 以及 FPGA demonstration 的 verilog 以及對應題號的結報內容 + 心得

陳弘軽 － 負責 Circuits of Basic Question 1,以及 Question 1、4 的 verilog 以及對應題號的結報內容